Inteligencia Artificial, 22(63) (2019), 16-38
doi: 10.4114/intartif.vol22iss63pp16-38

INTELIGENCIA ARTIFICIAL

http://journal.iberamia.org/

# Stereo Matching through Squeeze Deep Neural Networks

Gabriel D. Caffaratti[1,2,A], Martin G. Marchetta[1,B], and Raymundo Q. Forradellas[1,C]

[1]Laboratorio de Sistemas Inteligentes (LABSIN), Facultad de Ingeniería - Universidad Nacional de Cuyo, Centro Universitario, Mendoza, Argentina
[2]CONICET

[A]gabriel.caffaratti@ingenieria.uncuyo.edu.ar
[B]martin.marchetta@ingenieria.uncuyo.edu.ar
[C]kike@uncu.edu.ar

**Abstract.** Visual depth recognition through Stereo Matching is an active field of research due to the numerous applications in robotics, autonomous driving, user interfaces, etc. Multiple techniques have been developed in the last two decades to achieve accurate disparity maps in short time. With the arrival of Deep Leaning architectures, different fields of Artificial Vision, but mainly on image recognition, have achieved a great progress due to their easier training capabilities and reduction of parameters. This type of networks brought the attention of the Stereo Matching researchers who successfully applied the same concept to generate disparity maps. Even though multiple approaches have been taken towards the minimization of the execution time and errors in the results, most of the time the number of parameters of the networks is neither taken into consideration nor optimized. Inspired on the Squeeze-Nets developed for image recognition, we developed a Stereo Matching Squeeze neural network architecture capable of providing disparity maps with a highly reduced network size without a significant impact on quality and execution time compared with state of the art architectures. In addition, with the purpose of improving the quality of the solution and get solutions closer to real time, an extra refinement module is proposed and several tests are performed using different input size reductions.

**Keywords**: Stereo Matching, Deep Learning, Squeeze Nets, Artificial Intelligence, Artificial Vision, Disparity Maps.

## 1 Introduction

Stereo Matching is a research field inspired in human capabilities, in particular the stereopsis which is the ability of gathering depth information from the pair of images retrieved by the human binocular vision. Getting this information is essential to make decisions in different applications which interact with the world, like robotic object manipulation, unmanned vehicles navigation, security systems, user interfaces, etc. Since Hannah[1] and Marr et al.[2] proposed the matching of two images to extract stereo information, a number of techniques were developed to achieve this goal. As these Stereo Matching techniques started showing similarities, Scharstein et al.[3] developed a taxonomy that precisely defined common steps on them, specified their goals and instructed the process to measure them. Since then, most of the efforts have been focused on measuring how accurate were the disparity maps obtained in the matching cost calculation and post-processing steps, and how fast they were built.

**ISSN: 1137-3601 (print), 1988-3064 (on-line)**
**©IBERAMIA and the authors**

Artificial Intelligence, and in particular Machine Learning, offer different techniques to solve complex problems through the training of different models. During training, these models learn to recognize different patterns in the data to perform a classification of the input provided. This represents an advantage over standard none-trainable programmatic techniques which can only recognize static defined patterns to perform classification. In other words, the analysis of the input data is performed automatically during training based on the expected results of the training set, rather than having to recognize them manually with the problem of the misinterpretation of the data or the omission of unseen patterns. Within this field, a popular technique was the MultiLayer Perceptron (MLP) developed by Rumelhart et al.[4], a trainable artificial neural network capable of learning models through the adjustment of the weights connected the different layers of neurons through the Back-Propagation algorithm. Even though these networks were widely applied after their appearance, one of their main problems was their lack of scalability as a product of the exponential growth of the number of weights when there is a big number of inputs and outputs. A different type of networks called Convolutional Neural Networks (CNN), a specific technique of Deep Learning, was also trained through the back-propagation algorithm [5]. However, CNN started to be widely adopted only when Hinton, G. [6] shown how to train Deep Network layers independently by tuning the back-propagation algorithm. Thanks to these improvements, CNN technique offered a much scalable version of a trainable neural network.

Multiple CNN architectures have been proposed due to their simple and flexible training mechanism. In addition, frameworks like Torch[7], Tensorflow[8] or Theano[9] simplified their construction, training and test. In particular, one of the benefits of using CNN appeared when they were applied in image recognition problems, outperforming all the state of the art techniques [10], and currently surpassing the human performance[11]. Due to their success in image classification problems, CNN were also applied recently by Zbontar et al.[12] for the disparity cost calculation, bringing CNN architectures to the Stereo Matching field for the first time.

Different challenges have been presented for image classification[13] and disparity map generation[14] aiming at reducing the error rate and execution time. The size of the network in terms of number of parameters is very important, because it affects the computational cost for training and execution, and also because several applications require remote updates of the trained architectures, presenting in some cases connectivity restrictions and making the size of the network an important feature to optimize. Iandola et al.[15] proposed a CNN architecture called Squeeze Nets for image recognition which decreases the number of parameters to train and store, thus reducing the size of the network significantly. Inspired on that work, we developed a squeeze-network-based model for the generation of disparity maps for Stereo Matching, which reduces the network size in storage, while also maintaining the runtime memory, execution time and quality of the results.

Due to the importance of high quality solutions generated close to real time, different additions were proposed. First, an extra refinement module based on well known morphological filters which helps to improve the solution quality depending on the system configuration with low execution time cost. Second, multiple experiments are performed using different input reductions, decreasing the execution time and making the systems suitable for real time applications with low impact on the quality. Both additions makes our solution an alternative to be used in embedded devices with less memory and resources. In addition with the usage of a Squeeze Net architecture, the reduction of the parameters of the network decreases the communication time in remote systems were the network weights needs to be replaced due to a better trained model or any other reason.

This paper is composed by a review of the state of the art techniques presented in Stereo Matching (Section 2), an explanation of our proposed architecture and extra refinement module (Section 3), experiments performed based on a case of study (Section 4) and the conclusions and future work proposals (Section 5).

## 2 Literature Review

The construction of disparity maps consist in calculating the distance between various points or sections in a scene according to the position of the cameras. This means the disparity map defines a sort of depth image where the shape of different objects can be presented in different colors or gray-scale according to how close are these to the cameras. This task has several complexities due to the nature of the variable characteristics of the pictures and the ambiguous information they contain. Researchers have dealt with these ambiguities by making different assumptions of the images or data contained on it. The first pair of assumptions taken were uniqueness and continuity. Uniqueness establishes that each item of each image must be assigned with at most one disparity value. This condition relies on the assumption that an item correspond to something that has a unique physical position. Continuity states that disparity varies smoothly except on object boundaries where there are depth discontinuities[2]. Another important assumption is the epipolar rectification of the images which reduces the matching process to one dimension, or in other words a matching calculate over an horizontal line. Based on these assumptions stereo matching techniques were able to proceed with the matching of objects. However, we are far away from resolving all the different problems in the topic. Other important problems in stereo matching are the occlusions, textureless or repetitive texture surfaces, shape of the objects, differences in the intensities or noise in the images among others.

At side of the problems presented above, researchers found a number of mechanisms to retrieve dense disparity maps which can be divided in two groups: the ones detached of CNN and the ones based on these type of architectures.

## 2.1 CNN detached Stereo Matching techniques

Since the taxonomy proposed in [3] multiple techniques were proposed for the retrieval of dept information. These techniques can be categorized in two main groups based on the way the disparity map is calculated. The solutions that calculates the matching cost comparing a windows of neighbor pixels to gradually build disparity maps are considered local methods. Opposite to this, the solutions that retrieve a complete map and iteratively optimize it are considered global methods.

An extensive survey of local and global stereo matching algorithms where different comparisons and measures are made can be found in Hamzah et al.[16] work.

## 2.2 CNN based Stereo Matching techniques

CNN architectures marked a huge improvement in Artificial Vision areas. Particularly, this kind of artificial networks brought the attention of researchers when the Alexnet created by Krizhevsky et al.[10] reduced more than 10% the error rate on image classification problems reaching a 15.3%. Such achievement caused a revolution in this research field, having multiple CNN architectures proposed in the last five years. Zeiler et al.[17] was able to tune the hyperparameters of AlexNet creating the ZFNet and obtaining a 14.8% error rate. Later on GoogLeNet architecture by Russakovsky et al.[18] introduced the concept of inception modules enlarging the number of layers of the net with very small convolutional layers obtaining a 6.61% error and a considerable reduction of the parameters. An impressive improvement of the accuracy was achieved by He et al.[11] proposing the addition of residual links between layers in the ResNet achieving an error rate of 3.58%. A parameter reduction approach presented by Iandola et al.[15] decreased the network parameters size by 50x obtaining the same error rate as AlexNet.

All the previously mentioned networks were applied to the recognition of images with great success, however they had potential for other Artificial Vision problems like Stereo Matching. Zbontar et al.[12] worked on this idea inspired by the popularity of CNN and Mei et al.[19] work. He proposed a CNN based cost matching architecture with a series of post-processing steps described in Mei et al. paper obtaining an error rate of 2.63% on their accurate network version. It worth to mention a better implemented architecture designed by Shaked et al.[20] combining Zbontar et al.[12] suggested network with the residual networks proposed by He et al.[11]. In addition, Yang et al.[21] used CNN to perform not only the cost calculation but also to learn the smoothness constraint. Our work is highly inspired on [11, 12, 19] papers combined with the proposed Squeeze nets proposed in [15]. Since Zbontar et al.[12] network provided reproducible results in our tests and due to a closer similarity of their architecture with our solution, compared with other solutions in the field of stereo matching, our results are later compared with this work in Section 4.

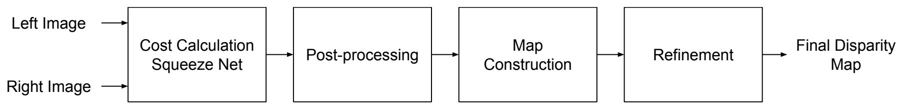

Figure 1: Four step proposed Stereo Matching system

### 2.3 Post-processing algorithms

In many occasions, the disparity maps obtained from the matching cost calculation process can be inaccurate, have occluded areas or unmatched sections. In order to improve their quality, different post-processing algorithms have been developed. An adaptive window algorithm based on the color similarity is proposed as cross-based cost aggregation by Ke Zhang et al.[22]. The method suggested by Hirschmüller[23] called semi-global matching (SGM) improves the smoothness term in the energy functions performing fast approximations of the neighbor pixels using Mutual Information in multiple directions. Another interesting post-processing method is proposed by Yao et al.[24] where the left and right based disparity maps are interpolated to get a better approximation of the mismatching areas. It worth to mention Williem et al.[25] who used a CNN network to increase the accuracy under the cost aggregation module. In addition, the disparity map results can be improved performing a quadratic interpolation of neighbor pixels as suggested by Miclea et al.[26].

## 3 Proposed System

Our model is composed of four steps as depicted the figure 1, following the common stereo matching taxonomy suggested by Scharstein et al.[3]. The system performs a cost calculation through a CNN extracting features of the left and right images individually and checking their similarity on a final layer, then a well known post-processing algorithm is applied in a cost aggregation module, then a disparity map is constructed based on the matching costs from the previous steps, and finally different known interpolation and image refinement algorithms are executed to obtain an optimized dense disparity map. Our main contribution can be found in the cost calculation module where a modified Squeeze Net architecture[15] is used to build a raw disparity map.

### 3.1 Cost Calculation module

The cost calculation of a common stereo matching algorithm is basically the comparison of pixels from different images. We implemented this functionality combining a CNN architecture and refinement algorithms as described below.

#### 3.1.1 Network Architecture

The proposed deep network architecture for cost calculation first processes the pair of images separately, executing two passes on the same layers, and then joins the results in a final layer, as depicted in the figure 2. The cost calculation network provides a raw disparity map, in the form of a 3D matrix where each element $(i, j, k)$ is the matching cost of pixel $(i, j)$ for disparity $k$.

Each image passes through a set of layers that generate feature maps. The feature maps obtained from each image are then fed into a similarity calculation layer at the end. The weights of the feature extraction layers are shared at the time of processing both the left and right images. The last layer performs a dot product between the feature maps of the left and right images, which were previously normalized. The normalization and dot product steps are equivalent to the cosine similarity measure which is used to retrieve a cost cube of a raw disparity map based on the similarity of the feature maps.

The first three layers are a Convolutional layer, a ReLU activation layer and a Pooling layer. The parameters $< K, S, P >$ shown in figure 2 are the *kernel size*, *stride* and *padding* of the module. $FM$ represents the number of features to be generated by the convolutional layer. The fire modules are

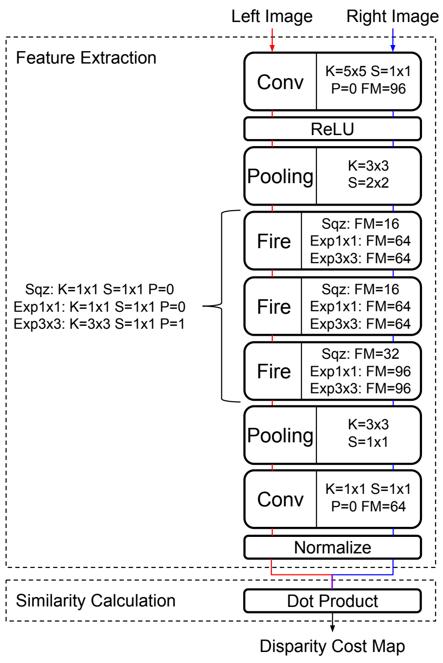

Figure 2: Cost Calculation network architecture. The left and right images are passed through the same feature extraction layers, and both results are then processed in the similarity calculation layer

composed of a set of layers. Each fire module shares the same $< K, S, P >$ parameters in the internal convolutional layers represented with $Sqz$, $Exp1x1$ and $Exp3x3$, but they differ in terms of feature maps.

### 3.1.2 Fire Module

The Fire module, depicted in the figure 3, is composed of two sequential steps with the purpose of squeezing the number of features received, and then expanding them in the next step, inspired by the Squeeze Net presented by Iandola et al.[15].

This model offers two advantages in terms of reduction of parameters. First, it reduces the number of input parameters to the convolutional layers. For example, an initial convolutional layer with 1x1 kernels (CL1x1) reduces by a factor of 9 the number of parameters of the layer when compared to a convolutional layer with 3x3 kernels (CL3x3), helping to decrease the size of the network. In addition, the number of features produced by the squeeze step is significantly reduced in contrast with other networks, generating a minimized input for the next CL3x3 of the expand module.

Second, in order to increase the number of features produced by the Fire module, a parallel CL1x1 is added, thus generating features maps that should otherwise be produced by the CL3x3. Consequently, the input required by both convolutional modules in the expand component to provide variety of feature maps can be reduced and so the number of parameters to be stored. It worths to mention that the CL1x1 of the expand module has padding equal to 1 in order to maintain the output feature map size equivalent to the CL3x3 one.

### 3.1.3 Cost Calculation

Since we assume that the images are rectified, their epipolar lines are completely horizontal and vertically coincident on both images. Unless they are occluded, the corresponding pixels can then be found in the same row, but differing by a certain disparity. Conversely, having the disparity corresponding to a pixel of one image, we can get the position of the matching pixel on the other image.

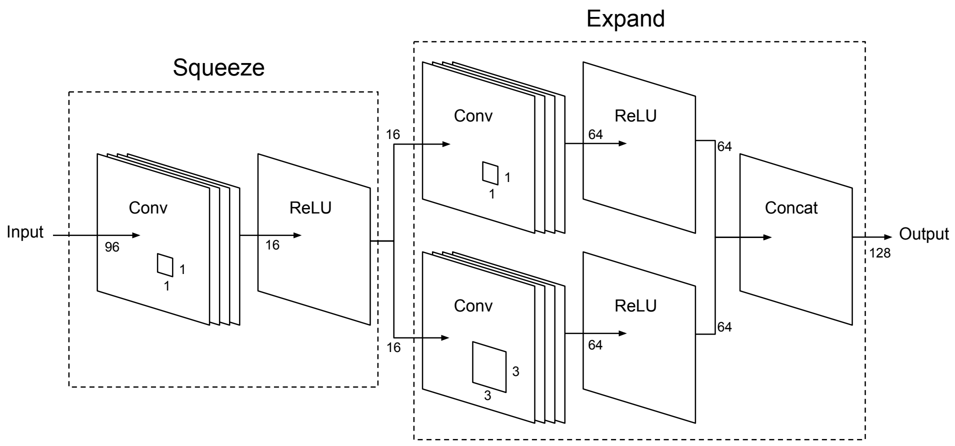

Figure 3: Fire Module layers composition. The number of feature maps in the output of the layers is illustrative and changes in each one of the fire modules of the architecture.

To train the proposed Squeeze Net we used the KITTI 2012 [27] or KITTI 2015 datasets[28, 29] which offers both left and right images rectified and a group of ground truth disparities. Taking these latter ones along with one of the images, we can create training patches with examples of correct and incorrect matches as suggested by Zbontar et al.[12]. Each patch is a matrix containing pixel intensities. The negative examples are obtained by adding an offset to the disparity provided by the ground truth, and getting the patch from the new position. Also we augmented the training samples by performing a series of transformations on the patches like rotation, brightness and saturation level modifications, among others. In this way we can train the network with positive and negative matching examples for each position and disparity.

Our cost function is described in the equation 1, where $P^L$ and $P^R$ are the left and right patches, $\mathbf{p}$ is the position of the center of the matrix representing the patch, $d$ is the horizontal displacement and $f$ is the network output, that measures the similarity of the patches, being zero when they match exactly.

$$C_{\text{Squeeze-Net}}(\mathbf{p}, d) = f\big(P^L(\mathbf{p}), P^R(\mathbf{p} - d)\big) \tag{1}$$

Because the output size we want to produce is fixed, we need to calculate the patch size $ws_k$ to produce it. The patch size is calculated only once, and is defined by equation 2, being $k$ the number of the last convolutional or pooling layer, $<S,\ P,\ K>$ the module's *stride*, *padding* and *kernel size* respectively, and $ws_{i-1}$ the output size of the module. Notice that $ws_k$ is calculated iteratively, where variable $i$ is initialized to 1 and it is increased step by step up to the number of convolutional and pooling layers, accumulating its values. In other words, to obtain the network minimum input size, equivalent to the patch size, we need to go backwards from the last convolutional or pooling layer to the first one.

$$ws_i = \begin{cases} 1 & i = 1 \\ \big((ws_{i-1} - 1) * S_{k-i+1}\big) - (2 * P_{k-i+1}) + K_{k-i+1} & 1 < i \leq k \end{cases} \tag{2}$$

### 3.1.4 Network Training

During training, the network analyzes pairs of left/right patches, and the result is a measure of their similarity, as stated in equation 1. This means the lost function used performs a binary classification of the pair of patches as matching or not matching, converting the cost calculation step in a classification problem. The loss function to be minimized to adjust the networks parameters applies the hinge-loss to pairs of positive and negative matching samples, as defined by equation 3

$$L = max(0, margin + m^- - m^+) \tag{3}$$

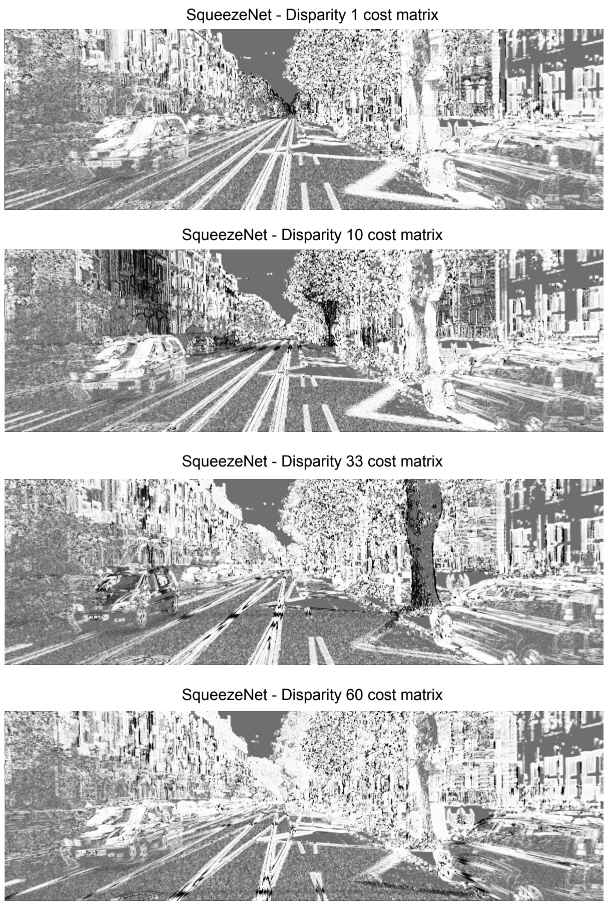
SqueezeNet - Disparity 1 cost matrix
SqueezeNet - Disparity 10 cost matrix
SqueezeNet - Disparity 33 cost matrix
SqueezeNet - Disparity 60 cost matrix

Figure 4: Different disparity matrices of the cost cube representing the cost calculated for each pixel.

where $m^-$ and $m^+$ are the results of the negative and positive matching examples, and $margin$ is the tolerance margin. When the similarity of the positive example is greater than the similarity of the negative one by a certain margin the loss will be zero. In our experiments the $margin$ used was 0.2.

### 3.1.5 Results Produced

The Cost Calculation module produces a cost cube composed of a series of matrices along the cube's depth, where each of these represents the cost of each pixel under a certain disparity. In case each matrix of the cost cube is graphed a distorted image is obtained where the pixels matching the matrix disparity appear in black as depicted in the figure 4.

In case the only module executed is the Cost Calculation module, the Disparity Map Generation process (explained in Section 3.3) needs to be executed in order to obtain the disparity map from the cost cube. This procedure and result is depicted in the figure 5.

## 3.2 Post-processing module

The cost calculation network in combination with a disparity map construction process (explained in Section 3.3) provides a representation of a raw disparity map, in the form of a 3D matrix where each element $(i, j, k)$ is the matching cost of pixel $(i, j)$ for disparity $k$. The quality of this map can be improved through a set of post-processing steps. In our work we only perform a Cross-based Cost Aggregation proposed by Zhang et al.[22] and Semi-Global Matching inspired by Hirschmüller et al.[23].

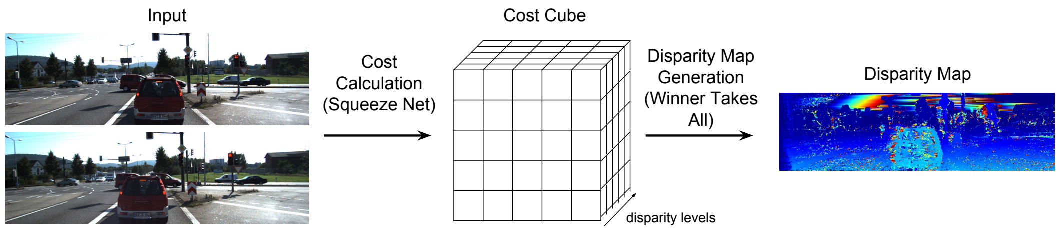
Input
Cost Calculation (Squeeze Net)
Cost Cube
disparity levels
Disparity Map Generation (Winner Takes All)
Disparity Map

Figure 5: Steps, subproducts and result obtained executing Cost Calculation module and Disparity Map Construction process only

### 3.2.1 Cross-based Cost Aggregation (CBCA)

This method generates an adaptive window starting from the pixel position. The window is generated by retrieving the pixels with similar intensity hoping they belong to the same object. First, for each pixel, a cross is built by checking if the difference between the intensities of that pixel and the up, down, left and right arms is under a certain threshold. The length of these arms is extended as long as the threshold is not exceeded or up to a certain arm length limit. Then, for each pixel of the vertical arm, a horizontal arm is added to the window following the same intensity and distance verification. This procedure is performed also for the matching image and then both windows are joined to construct a final window. The equation 4 depicts the criteria used to construct each up and down arm (vertical arm) and the equation 5 shows the same condition used to construct each left and right arms (horizontal arms) for each pixel of the vertical arm created in the previous equation.

$$Arm_V(\mathbf{p}) = \begin{cases} I(\mathbf{p}_{U|D}) & |I(\mathbf{p}) - I(\mathbf{p}_{U|D})| < cbca\_intensity \wedge ||\mathbf{p} - \mathbf{p}_{U|D}|| < cbca\_distance \\ None & otherwise \end{cases} \tag{4}$$

$$Arm_H(\mathbf{p}_{V_i}) = \begin{cases} I(\mathbf{p}_{L|R})_{V_i} & |I(\mathbf{p}) - I(\mathbf{p}_{L|R_{V_i}})| < cbca\_intensity \wedge ||\mathbf{p} - \mathbf{p}_{L|R_{V_i}}|| < cbca\_distance \\ None & otherwise \end{cases} \tag{5}$$

Where $\mathbf{p}$ is the position selected pixel to create region, $\mathbf{p}_{U|D}$ is the position of the pixel in the same vertical line of $\mathbf{p}$, $\mathbf{p}_{V_i}$ is the position of one of the pixels in the vertical arm $Arm_V$, $\mathbf{p}_{L|R_{V_i}}$ is the position of the pixel in the same horizontal line of the vertical arm pixel $\mathbf{p}_{V_i}$, $cbca\_intensity$ and $cbca\_distance$ are the intensity and distance thresholds respectively.

The support region of the selected pixel is built by joining all the horizontal arms of the vertical arm as depicted in the equation 6.

$$R^L(\mathbf{p}) = \bigcup_{\mathbf{p}_{V_i}}^{\mathbf{p}_{V_i} \in Arm_V} Arm_H(\mathbf{p}_{V_i}) \tag{6}$$

Then a combined support region is created by joining the left and right images' regions as shown in the equation 7.

$$U_d(\mathbf{p}) = \{\mathbf{q} | \mathbf{q} \in R^L(\mathbf{p}), \mathbf{q} - d \in R^R(\mathbf{p} - d)\} \tag{7}$$

were $R^L$ and $R^R$ are the left and right images regions for the pixel position $\mathbf{p}$. Finally, an average of the cost of all the pixels over the combined region is obtained and assigned to the pixel position used at the beginning of the process. This can be seen in the equation 8.

$$C_{CBCA}(\mathbf{p}) = \frac{1}{|U_d(\mathbf{p})|} \sum_{\mathbf{q} \in U_d(\mathbf{p})} C_r(\mathbf{q}, d) \tag{8}$$

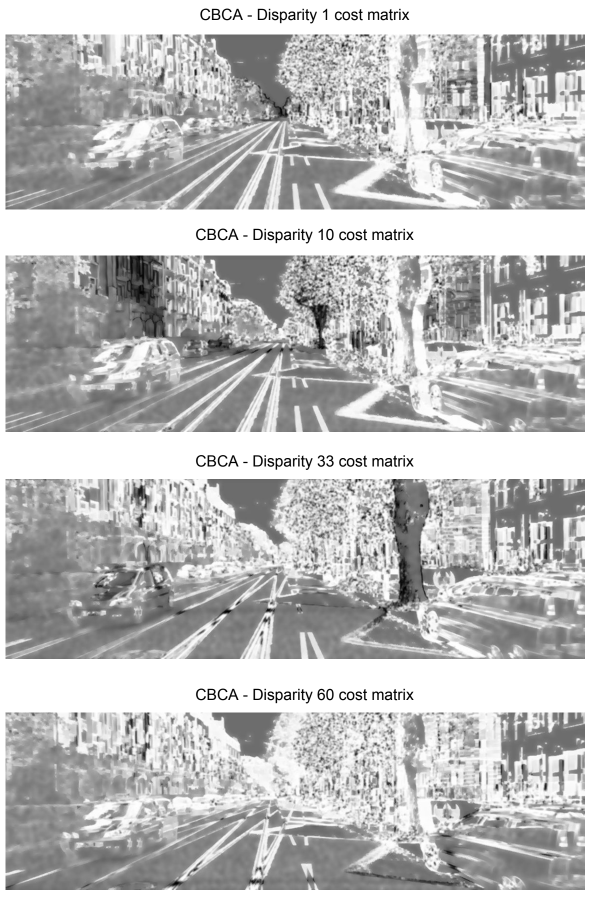
CBCA - Disparity 1 cost matrix
CBCA - Disparity 10 cost matrix
CBCA - Disparity 33 cost matrix
CBCA - Disparity 60 cost matrix

Figure 6: Different matrices of the cost cube representing the cost aggregation for each disparity after the CBCA procedure.

This process can be iterated different number of times since the support regions overlap between them producing different results on each iterations. It worth to mention the input of this procedure is each matrix of the cost cube produced by the Cost Calculation module or any previous procedure of the post-processing module. Then the final product of the CBCA is an improved cost cube where the matching pixels of each disparity are better defined with less noise as depicted in the figure 6.

### 3.2.2 Semi-global Matching (SGM)

An important refinement of the disparity map is related to smoothness. Since objects usually have similar disparities, intuitively we can say that differences in the disparity of neighboring pixels should be penalized. In general, the penalty applied increases according to how strong the difference between disparities is. We defined the local neighborhood of a pixel by moving 1 pixel away of if in the up, down, left and right directions, following the suggestion in [12]. If we call $\mathbf{r}$ the vector of the directions where the pixels $\mathbf{q}$ are found, a set of adjustments $C_{\mathbf{r}}(\mathbf{p}, d)$ for the matching cost $C_{\mathbf{r}(\mathbf{p},d)}$ computed by the CNN are calculated through equation 9 (and later averaged, as described below).

$$\begin{aligned} C_{\mathbf{r}}(\mathbf{p}, d) = C_D(\mathbf{p}, d) - \min_i C_{\mathbf{r}}(\mathbf{p}-\mathbf{r}, i) + \min\Big(C_{\mathbf{r}}(\mathbf{p}-\mathbf{r}, d)\boldsymbol{,} \\ C_{\mathbf{r}}(\mathbf{p}-\mathbf{r}, d-1) + P_1\boldsymbol{,} C_{\mathbf{r}}(\mathbf{p}-\mathbf{r}, d+1) + P_1\boldsymbol{,} \min_i C_{\mathbf{r}}(\mathbf{p}-\mathbf{r}, i) + P_2\Big) \end{aligned} \tag{9}$$

Here $C_D$ is the matching cost calculated by the CNN, $P_1$ is the penalty when the difference between the disparity $D_{\mathbf{p}}$ and $D_{\mathbf{q}}$ of pixels **p** and **q** in the local neighborhood is 1. $P_2$ is the penalty when that difference is higher than 1, and $i$ are the valid disparities. The second term of the equation compensates for cases where the values of the third term of the equation grow too large, for smoothing special cases (e.g. occluded or mismatching pixels).

The penalty values $P_1$ and $P_2$ varies according to disparity of the pixel compared with neighbors in edges of objects. Therefore, penalties are lower when pixels are detected in borders. The parameters are defined as follows:

$$D_1 = |I^L(\mathbf{p}) - I^L(\mathbf{p}-\mathbf{r})| \qquad D_2 = |I^R(\mathbf{p}-\mathbf{d}) - I^R(\mathbf{p}-\mathbf{d}-\mathbf{r})|$$

$$\begin{array}{lll} P_1 = P_1, & P_2 = P_2, & \text{if } D_1 < D_{sgm} \wedge D_2 < D_{sgm}; \\ P_1 = P_1/Q_2, & P_2 = P_2/Q_2, & \text{if } D_1 \geq D_{sgm} \wedge D_2 \geq D_{sgm}; \\ P_1 = P_1/Q_1, & P_2 = P_2/Q_1, & \text{otherwise} \end{array} \tag{10}$$

where $I$ is the pixel intensity. In case the disparity shows a big discontinuity in the disparity map, i.e. when $D_1$ and $D_2$ are equal or bigger than a certain $D_{sgm}$, the penalty is reduced by a large factor $Q_2$ as it is assumed that it is a steep border. In case just $D_1$ or $D_2$ meets this condition, the border is not that steep so the penalty is reduced by a smaller factor $Q_1$. Otherwise the case is not a border. In case of vertical directions a different factor $V$ is used to reduce $P_1$ as the disparities changes shown in ground truth images are more frequently vertical. After computing all these values for each direction, the final smoothed cost is an average of the results obtained (equation 11).

$$C_{\text{SGM}}(\mathbf{p}, d) = \frac{1}{4} \sum_{\mathbf{r}} C_{\mathbf{r}}(\mathbf{p}, d) \tag{11}$$

Similar to the CBCA process described before, the input of the SGM process are the different matrices of the cost cube produced by the Cost Calculation module or any previous post-processing procedure. Then, the product obtained after executing the SGM process is a refined cost matrix for each disparity of the cost cube with better defined matching pixels as depicted in the figure 7.

### 3.2.3 Results Produced

Once the different post-processing procedures are executed the result is an improved cost cube where the matching cost for each disparity is better defined. Then, this cube can be provided to the Map Generation module (explained in Section 3.3) to produce a disparity map with probably less errors than another one where no post-processing module was executed. The subproducts and complete process are depicted in the figure 8.

## 3.3 Map Construction

The disparity map is constructed by gathering the minimum cost from the cube of matching costs for each pixel position. This strategy is called winner-takes-all and is defined in the equation 12.

$$D(\mathbf{p}) = \arg\min_{d} C(\mathbf{p}, d) \tag{12}$$

## 3.4 Refinement module

Once the disparity map is constructed, problems like mismatching or occluded pixels can reduce its accuracy. This can be improved through different interpolation and refinement steps. These steps were inspired by Mei et al.[19] work. The execution order is depicted in figure 9.

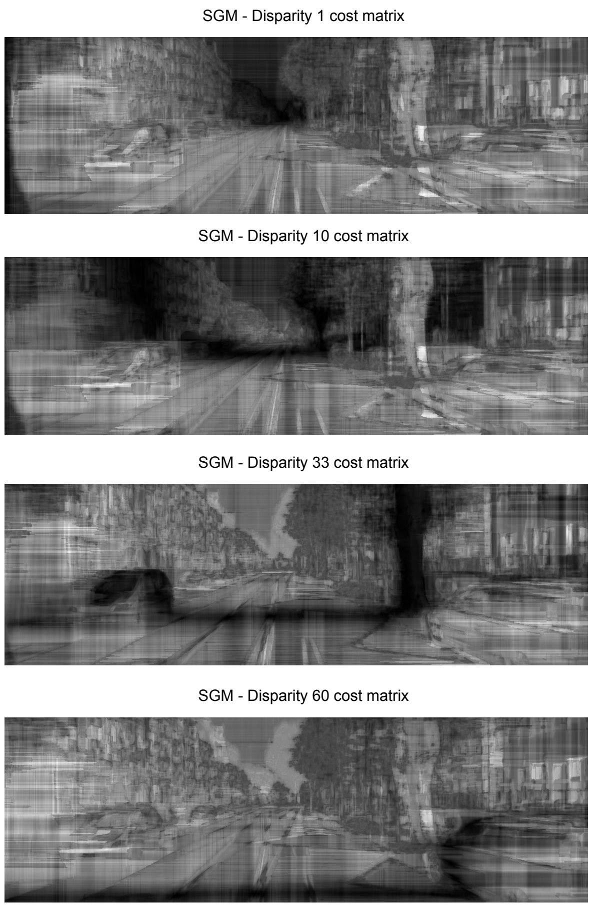
SGM - Disparity 1 cost matrix
SGM - Disparity 10 cost matrix
SGM - Disparity 33 cost matrix
SGM - Disparity 60 cost matrix

Figure 7: Different matrices of the cost cube representing the cost aggregation for each disparity after the SGM procedure.

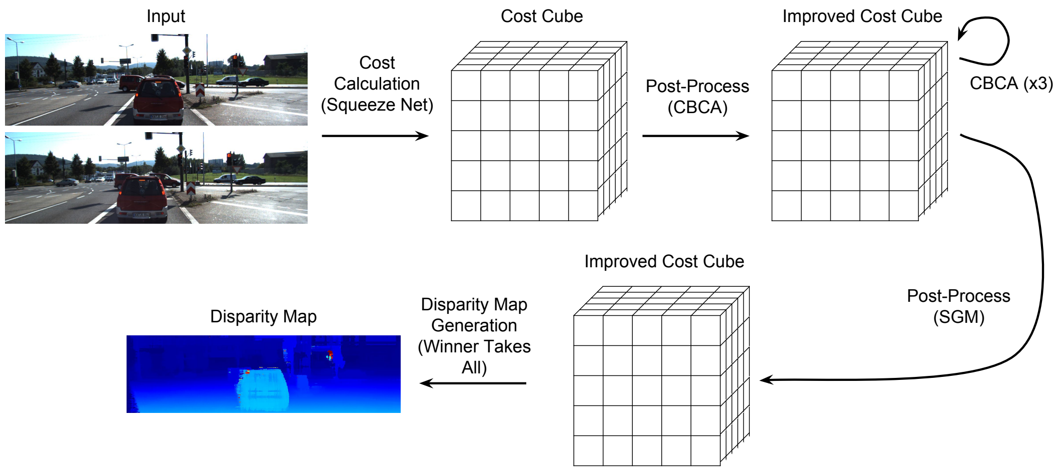

Figure 8: Steps, subproducts and result obtained executing Cost Calculation module followed by four iterations of CBCA, then the SGM process was executed and finalizing with a Disparity Map Generation process

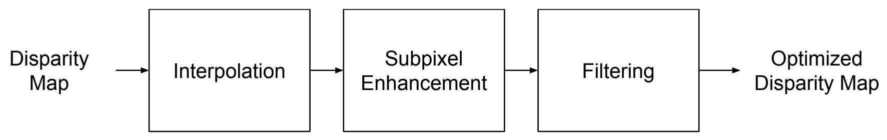

Figure 9: Refinement steps performed after the map construction module

### 3.4.1 Interpolation

The disparity map can be calculated either using the left image as reference or the right image. Changing the reference image produces different maps since the occluded pixels are different on each one. Having both disparity maps can help us to determine which are those occluded object by comparing the disparities of a matching pixel. If the disparities match (the absolute difference is less than one), we can consider them as correct. If the disparity of the pixel in one map matches the disparity of a pixel in the other map other than the corresponding one, we consider it as incorrect. If the disparity does not match any other disparity in the other map, it is an occluded pixel.

The occluded pixels disparity is obtained by looking at the nearest correct pixel at the left. For mismatching pixels we look for a disparity value as the median of sixteen directions pixels around them.

### 3.4.2 Sub-pixel Enhancement

In this step we use the SGM cost of pixel $p$ and disparity $d$ and its closest "disparity" neighbors to get a smoothing subtraction term, to improve the result, as shown in equation 13.

$$D(\mathbf{p}, d) = D(\mathbf{p}, d) - \frac{C_{SGM}(\mathbf{p}, d+1) - C_{SGM}(\mathbf{p}, d-1)}{2\Big(C_{SGM}(\mathbf{p}, d+1) - 2C_{SGM}(\mathbf{p}, d) + C_{SGM}(\mathbf{p}, d-1)\Big)} \tag{13}$$

### 3.4.3 Filtering

This module applies a median filter with a 5x5 kernel followed by a bilateral filter, for the purpose of smoothing disparity changes without affecting the edges.

### 3.4.4 Results Produced

After the disparity map is constructed and the different refinement steps are executed the result is a filtered and smoothed disparity map as depicted in the figure 10.

## 3.5 Erode-Dilate module

With the purpose of improving the accuracy of the system, an extra module is proposed conformed by a binary mask generator and two morphological filters which can be applied to a disparity map. These morphological filters are the well known erosion and dilation processes which are basically conformed by a kernel of a certain size which is passed through all the disparity map mask. This module is depicted in the figure 11.

Once both filters are applied, the isolated pixels over a certain threshold surrounded by other pixels under the threshold are marked with the lowest disparity, as background, removing the disparity noise in early steps of the system. Taking into consideration that the disparities of the background of an image are usually smooth, this module tends to improve the accuracy of the generated disparity map by removing isolated foreground disparity values.

### 3.5.1 Mask Generation

A simple mask is generated by checking each value of the disparity map against a certain threshold $t$ as shown in the equation 14.

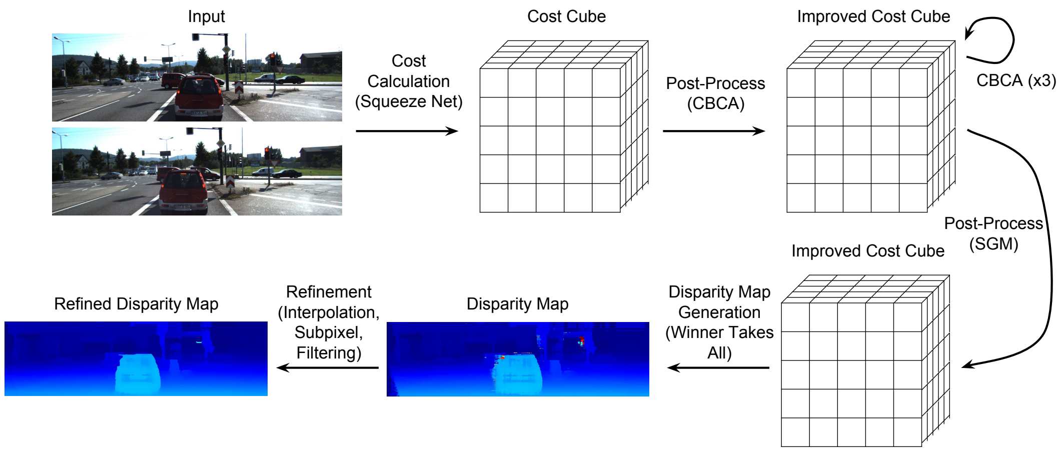

Figure 10: Steps, subproducts and result obtained executing Cost Calculation module followed by four iterations of CBCA, then the SGM process was executed and finalizing with a Disparity Map Construction and a full Refinement process

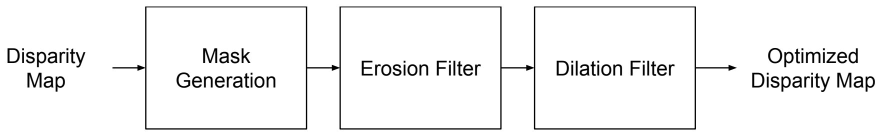

Figure 11: Steps performed in the Erode-Dilate module after the map construction or refinement module

$$D_{mask}(\mathbf{p}, d) = \begin{cases} d & d \geq t \\ 0 & otherwise \end{cases} \tag{14}$$

Every value under the threshold is considered as background, otherwise as foreground. In this way the background disparities are set to zero, generating a binary map composed by background (zero disparity) and foreground (any disparity over the threshold) values.

### 3.5.2 Erosion Filter

This step is composed by a binary erosion of the image performed by a binary kernel passed over each position of the disparity map obtained from the previous step. The kernel is defined with some of its values set as one. Taking the disparity map mask as binary image, the kernel is compared on each location of the binary image to verify if the kernel values matches the region being compared. In case region doesn't match, the center position of the region is set to zero, otherwise this value is kept unchanged as denoted in the equation 15.

$$D_{eroded}(\mathbf{p}, d) = D_{mask}(\mathbf{p}, d) \ominus K = \{d \in D_{mask} \mid K_{\mathbf{p}} \subseteq D_{mask}(\mathbf{p}, d)\} \tag{15}$$

Here $K$ is the kernel, $K_{\mathbf{p}}$ is the kernel at the position $\mathbf{p}$ and $d$ is the disparity at the center of that position.

The erosion process has the effect of removing the isolated disparities of the disparity map, but also makes the shapes in the conformed binary image thinner. This undesired effect can be mitigated applying a dilation filter over the result.

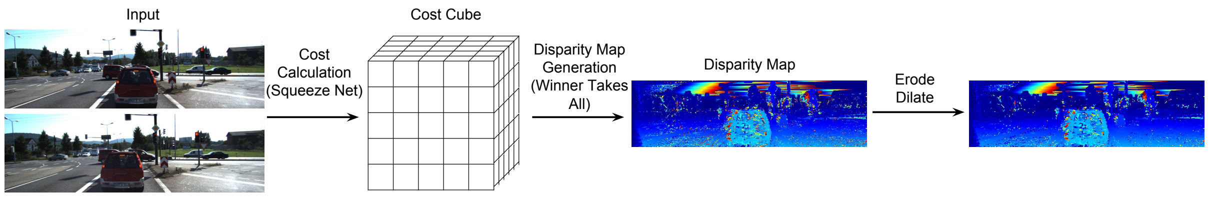
Input
Cost Calculation (Squeeze Net)
Cost Cube
Disparity Map Generation (Winner Takes All)
Disparity Map
Erode Dilate

Figure 12: Steps, subproducts and result obtained executing Cost Calculation module followed by a Disparity Map Construction and the Erode-Dilate refinement process

### 3.5.3 Dilation Filter

As mentioned in the previous step, the execution of a dilation filter is required to mitigate some undesired effects of the erosion filter. This process has the same behavior of the erosion filter except that performs an union with the region being compared if any of the of the values set in the kernel matches. This can be seen in the equation 16.

$$D_{dilated}(\mathbf{p}, d) = D_{eroded}(\mathbf{p}, d) \oplus K = \{d \in D_{mask} \mid K_{\mathbf{p}} \cup D_{eroded}(\mathbf{p}, d) \wedge (K_{\mathbf{p}} \cap D_{eroded}(\mathbf{p}, d)) \neq \varnothing\} \quad (16)$$

As in in the equation 15, here $K$ is the kernel, $K_{\mathbf{p}}$ is the kernel at the position $\mathbf{p}$ and $d$ is the disparity at the center of that position.

It's important to notice that the disparities added by the dilation process should be retrieved from $D_{mask}$ in order to obtain the thickening effect of the dilation process on the disparity map shapes but respecting the original disparities removed by the erosion process.

### 3.5.4 Results Produced

Similar to the Refinement module, the Erode-Dilate module is executed after the Disparity Map Construction module. The result is a less noisy disparity map where the isolated errors in the foreground were removed as depicted in the figure 12.

## 4 Case Study

In this section we present details of the environment where the different algorithms were executed, the training specifications and the system setup. The results obtained are shown in terms of parameters reduction, error rate, execution time and memory consumption. The experiments were performed on an AMD Ryzen 1700 CPU with 32 GB DDR–4 2400 MHz ram and a NVidia Titan Xp GPU. The source code of the project were all the experiments were performed can be found in a git repository[1].

### 4.0.1 Training set

We used the 2012 KITTI dataset [27] and 2015 KITTI dataset [28, 29] composed by 194 and 200 training pairs of images of 1240 x 376 pixels respectively. These images have a maximum of 228 disparity levels. The dataset provides around 30% of the image disparities measured with laser scanners. Leaving 40 images for test, the remaining 154 and 160 images made a training set of around 19 million positions with measured disparity on each dataset. Since we get positive and negatives examples as mentioned in Section 3.1.3, the number of training examples (sampled patches) is more than 38 million. Each sample is subject to several image transformations in order to provide different samples on each epoch. As a product of the CNN layers' kernels and padding the window size of our patches is 9x9.

[1]https://github.com/labsin-uncuyo/gdc-mc-cnn

#### 4.0.2 Learning parameters

These training samples were provided in batches of 128 samples during 15 epochs. We used a learning rate of 0.05 which is decreased after epoch 11 by a factor of 10. The parameters used for image post-processing are the same used in Zbontar et al.[12] fast architecture.

#### 4.0.3 Post-processing module configurations

In order to perform a fair comparison between Squeeze Net and Zbontar et al.[12] fast architecture solutions, the post-processing module we used executes only the SGM process, mentioned in Section 3.2 when using Kitti 2012 dataset. However, we included the CBCA process in our tests using Kitti 2015 to see how much affected are the memory, execution time and accuracy under this configuration. Also, the Zbontar et al.[12] accurate architecture uses both CBCA and SGM processes for post-processing. The different configurations are depicted in the table 1.

Table 1: Post-processing module configurations used over the different architectures and datasets

| Configuration | CBCA (before SGM) | SGM | CBCA (after SGM) |
|---|---|---|---|
| ZBontar Fast (Kitti 2012) | | • | |
| ZBontar Fast (Kitti 2015) | • | • | • |
| ZBontar Accurate | • | • | • |
| Squeeze Net (Kitti 2012) | | • | |
| Squeeze Net (Kitti 2015) | • | • | • |

#### 4.0.4 Cost calculation parameters

The proposed Squeeze Deep Neural network considerably decreased the number of parameters in relation with the other two networks. This result is the product of the reduction of features in the squeeze layer of the fire module that are fed into the expansion CL3x3 module and the expansion of features through a parallel CL1x1. A comparison of the network size, the parameters involved and the reduction rate is shown in the table 2.

Table 2: Cost Calculation Deep Neural network sizes and parameters

| Network | Size (KB) | # Parameters | Reduction |
|---|---|---|---|
| Zbontar Accurate | 2,534 | 648,592 | 87.81% |
| Zbontar Fast | 440 | 112,564 | 29.77% |
| **Squeeze Net** | **309** | **79,040** | - |

#### 4.0.5 System accuracy and execution time

The system was executed in two instances: with post-processing and refinement, and without post-processing. Table 3 shows the different results obtained as an average over the 40 testing images.

Table 3: System accuracy and execution time using Kitti 2012 dataset

| | Error | | Execution time | |
|---|---|---|---|---|
| Network | CNN + REF | Full method | CNN + REF | Full method |
| Zbontar Accurate | 6.03% | 2.54% | 33.24 sec | 33.84 sec |
| Zbontar Fast | 8.39% | 2.93% | 0.33 sec | 0.65 sec |
| Squeeze Net | 11.87% | 3.69% | 0.58 sec | 0.89 sec |

Even through the error is 6% higher in case the system is executed without post-processing, the degradation of the quality including this module is below 1.5% even compared with the accurate architecture.

The execution time of the network is only comparable with the fast architecture and is over 38 times faster than the accurate architecture. The resulting disparity maps of each model, including post-processing and refinement steps, are shown in figure 13.

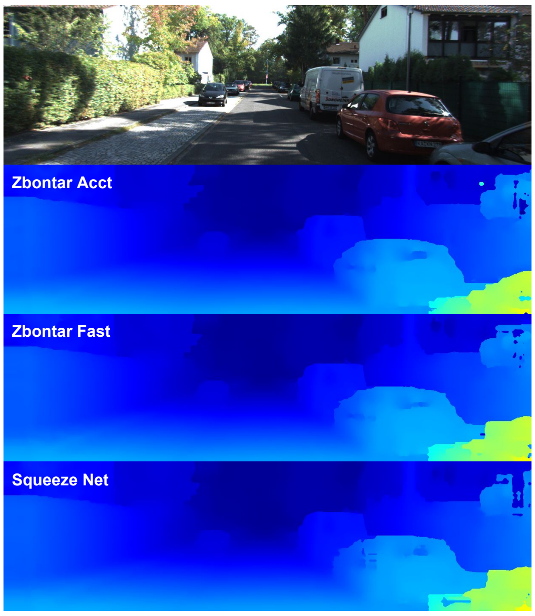

Figure 13: Full method generated disparity maps

#### 4.0.6 GPU Memory consumption

There is a reduction of memory consumption by our solution, as compared to the fast architecture, making it feasible to use in smaller devices with less GPU resources as presented in table 4.

Table 4: Architecture GPU memory consumption comparison

| Network | Zbontar Accurate | Zbontar Fast | Squeeze Net |
|---|---|---|---|
| GPU Mem | ∼4200 MB | ∼2000 MB | ∼2000 MB |

#### 4.0.7 Reduction of the input images

In order to decrement the execution time of the stereo matching system and get closer to real time processing, different executions were performed using a reduced versions of the input images. At the end of system process, the output has a equivalent size of the input images. Since the ground truth is in the original size, a fair comparison of the results is performed by resizing the output to the original size, and then verifying all the measured points of the ground truth with the final disparity map obtained. The experiments were performed with different grades of input reductions over two of the previously described systems. The first one is the ZBontar et al. Fast architecture and the second one is our proposed Squeeze Net architecture. Both systems share the same post-processing and refinement modules as they were described in the previous sections, differing only in the cost calculation module composed by different CNN network architectures. However different executions were performed removing some modules in order to determine how a reduced input was affecting in the different parts of the system. A table of the different system configurations is shown in the table 5.

The resizing of the inputs' dimensions on both networks shown a decrement of the execution time using both Kitti 2012 and 2015 datasets. The ZBontar et al. architecture best result reached a 4.36 times reduction factor in terms of execution time of the system running only the cost calculation and

Table 5: Configurations used to run the experiments with reduced input size

| Configuration | Cost Calculation | Post-Processing | Refinement |
|---|---|---|---|
| C1 | ● | | |
| C2 | ● | | ● |
| C3 | ● | ● | |
| C4 | ● | ● | ● |

postprocessing modules when an input of 50% size was used. In case of the Squeeze Net architecture there was a 3.39 times reduction factor executing the same modules with the same input size. Both systems were close to real time solutions having 0.15 and 0.26 seconds of execution time respectively. Similar but better results were obtained using Kitti 2015 dataset. Under the same configuration and input size the Zbontar et al. architecture best result reached 5.52 times reduction factor while the Squeeze Net architecture had a 4.14 times reduction factor. Also, both system got close to real time solutions having 0.17 and 0.28 seconds of execution time respectively. The rest of the results of the different executions are shown in the table 6 and table 7 for Kitti 2012 dataset, and table 8 and table 9 for Kitti 2015.

The accuracy error had a incremental factor in the cases the post-processing module was executed according with the amount of reduction performed in the input. However there was a reduction of the error in the cases where cost calculation only or cost calculation plus refinement was used. A complete list of the results obtained is shown in the table 10 and table 11 for the ZBontar et al. and Squeeze Net architectures respectively using Kitti 2012 dataset, and table 12 and table 13 using Kitti 2015 dataset.

The memory consumption was also taken into consideration after each execution, recording the reduction factor on every input reduction over the different configurations of both systems using Kitti 2012 and 2015 datasets. In both systems there was a higher memory saving compared with the reduction performed on the inputs reaching a factor of 2.93 and 2.33 for the ZBontar et al. and Squeeze Net architectures respectively with both datasets. The rest of the results of the different executions are shown in the table 14 and table 15 for Kitti 2012 dataset, and table 16 and table 17 for Kitti 2015 dataset.

Table 6: Execution times of the ZBontar Fast architecture using different input sizes on different modules over Kitti 2012 dataset

| Execution Time in Seconds (Reduction Factor) | | | | | | |
|---|---|---|---|---|---|---|
| | Input Size | | | | | |
| Conf. | 100% | 90% | 80% | 70% | 60% | 50% |
| C1 | 0.33 | 0.27 (1.22) | 0.22 (1.52) | 0.16 (2.14) | 0.14 (2.40) | 0.11 (3.06) |
| C2 | 0.34 | 0.28 (1.22) | 0.22 (1.57) | 0.17 (1.98) | 0.13 (2.55) | 0.11 (3.00) |
| C3 | 0.65 | 0.50 (1.30) | 0.38 (1.71) | 0.26 (2.53) | 0.21 (3.06) | 0.15 (4.36) |
| C4 | 0.67 | 0.52 (1.29) | 0.39 (1.70) | 0.30 (2.26) | 0.22 (2.09) | 0.16 (4.12) |

Table 7: Execution times of the Squeeze Net architecture using different input sizes on different modules over Kitti 2012 dataset

| Execution Time in Seconds (Reduction Factor) | | | | | | |
|---|---|---|---|---|---|---|
| | Input Size | | | | | |
| Conf. | 100% | 90% | 80% | 70% | 60% | 50% |
| C1 | 0.58 | 0.48 (1.21) | 0.41 (1.42) | 0.29 (2.01) | 0.22 (2.61) | 0.22 (2.68) |
| C2 | 0.59 | 0.49 (1.20) | 0.41 (1.45) | 0.34 (1.74) | 0.28 (2.15) | 0.23 (2.58) |
| C3 | 0.89 | 0.72 (1.25) | 0.57 (1.56) | 0.39 (2.30) | 0.29 (3.11) | 0.26 (3.39) |
| C4 | 0.91 | 0.73 (1.25) | 0.59 (1.55) | 0.46 (1.97) | 0.35 (2.56) | 0.29 (3.13) |

Table 8: Execution times of the ZBontar Fast architecture using different input sizes on different modules over Kitti 2015 dataset

| Execution Time in Seconds (Reduction Factor) | | | | | | |
|---|---|---|---|---|---|---|
| | Input Size | | | | | |
| Conf. | 100% | 90% | 80% | 70% | 60% | 50% |
| C1 | 0.33 | 0.28 (1.18) | 0.22 (1.53) | 0.16 (2.10) | 0.11 (2.96) | 0.11 (3.01) |
| C2 | 0.33 | 0.28 (1.21) | 0.22 (1.50) | 0.18 (1.89) | 0.13 (2.50) | 0.12 (2.90) |
| C3 | 0.92 | 0.72 (1.28) | 0.53 (1.73) | 0.35 (2.64) | 0.26 (3.49) | 0.17 (5.52) |
| C4 | 0.94 | 0.73 (1.29) | 0.54 (1.74) | 0.38 (2.45) | 0.27 (3.54) | 0.19 (5.01) |

Table 9: Execution times of the Squeeze Net architecture using different input sizes on different modules over Kitti 2015 dataset

| Execution Time in Seconds (Reduction Factor) | | | | | | |
|---|---|---|---|---|---|---|
| | Input Size | | | | | |
| Conf. | 100% | 90% | 80% | 70% | 60% | 50% |
| C1 | 0.59 | 0.49 (1.19) | 0.41 (1.44) | 0.29 (2.02) | 0.22 (2.72) | 0.23 (2.53) |
| C2 | 0.59 | 0.49 (1.20) | 0.41 (1.44) | 0.34 (1.72) | 0.28 (2.13) | 0.23 (2.59) |
| C3 | 1.17 | 0.92 (1.27) | 0.73 (1.62) | 0.48 (2.44) | 0.32 (3.66) | 0.28 (4.14) |
| C4 | 1.19 | 0.93 (1.27) | 0.73 (1.61) | 0.55 (2.14) | 0.41 (2.93) | 0.31 (3.85) |

Table 10: Accuracy errors of the ZBontar Fast architecture using different input sizes on different modules over Kitti 2012 dataset

| Accuracy Error (Increase Factor) | | | | | | |
|---|---|---|---|---|---|---|
| | Input Size | | | | | |
| Conf. | 100% | 90% | 80% | 70% | 60% | 50% |
| C1 | 15.33% | 14.89% (0.97) | 13.11% (0.86) | 11.59% (0.76) | 10.52% (0.69) | 9.96% (0.65) |
| C2 | 8.39% | 7.38% (0.88) | 6.59% (0.79) | 6.05% (0.72) | 6.17% (0.74) | 7.42% (0.88) |
| C3 | 4.00% | 4.09% (1.02) | 4.17% (1.04) | 4.36% (1.09) | 4.66% (1.16) | 5.25% (1.31) |
| C4 | 2.93% | 3.17% (1.08) | 3.40% (1.16) | 3.88% (1.32) | 4.78% (1.63) | 6.87% (2.34) |

Table 11: Accuracy errors of the Squeeze Net architecture using different input sizes on different modules over Kitti 2012 dataset

| Accuracy Error (Increase Factor) | | | | | | |
|---|---|---|---|---|---|---|
| | Input Size | | | | | |
| Conf. | 100% | 90% | 80% | 70% | 60% | 50% |
| C1 | 28.77% | 31.30% (1.09) | 28.60% (0.99) | 26.12% (0.91) | 24.27% (0.84) | 23.37% (0.81) |
| C2 | 11.87% | 11.21% (0.94) | 10.05% (0.85) | 9.29% (0.78) | 9.17% (0.77) | 10.15% (0.86) |
| C3 | 5.27% | 5.45% (1.03) | 5.77% (1.10) | 6.26% (1.19) | 7.00% (1.33) | 8.22% (1.56) |
| C4 | 3.69% | 4.10% (1.11) | 4.57% (1.24) | 5.36% (1.45) | 6.76% (1.83) | 9.51% (2.58) |

Table 12: Accuracy errors of the ZBontar Fast architecture using different input sizes on different modules over Kitti 2015 dataset

| Accuracy Error (Increase Factor) | | | | | | |
|---|---|---|---|---|---|---|
| | Input Size | | | | | |
| Conf. | 100% | 90% | 80% | 70% | 60% | 50% |
| C1 | 15.24% | 14.72% (0.97) | 13.22% (0.87) | 12.09% (0.79) | 13.87% (0.91) | 11.21% (0.74) |
| C2 | 8.17% | 7.44% (0.91) | 6.92% (0.85) | 6.61% (0.81) | 6.78% (0.83) | 7.67% (0.94) |
| C3 | 4.64% | 4.76% (1.03) | 4.91% (1.06) | 5.19% (1.12) | 5.72% (1.23) | 6.55% (1.41) |
| C4 | 4.04% | 4.30% (1.06) | 4.53% (1.12) | 4.98% (1.23) | 5.77% (1.43) | 7.41% (1.83) |

Table 13: Accuracy errors of the Squeeze Net architecture using different input sizes on different modules over Kitti 2015 dataset

| Accuracy Error (Increase Factor) | | | | | | |
|---|---|---|---|---|---|---|
| | Input Size | | | | | |
| Conf. | 100% | 90% | 80% | 70% | 60% | 50% |
| C1 | 28.734% | 29.78% (1.04) | 27.26% (0.95) | 25.13% (0.87) | 23.57% (0.82) | 21.96% (0.76) |
| C2 | 12.048% | 11.22% (0.93) | 10.23% (0.85) | 9.60% (0.80) | 9.50% (0.79) | 10.22% (0.85) |
| C3 | 5.991% | 6.27% (1.05) | 6.68% (1.11) | 7.27% (1.21) | 8.26% (1.38) | 9.83% (1.64) |
| C4 | 5.046% | 5.49% (1.09) | 5.98% (1.19) | 6.74% (1.34) | 7.95% (1.58) | 10.32% (2.04) |

Table 14: Memory usage of the ZBontar Fast architecture using different input sizes on different modules over Kitti 2012 dataset

| Memory Usage in Mb (Reduction Factor) | | | | | | |
|---|---|---|---|---|---|---|
| | Input Size | | | | | |
| Conf. | 100% | 90% | 80% | 70% | 60% | 50% |
| C1 | 2029 | 1617 (1.25) | 1283 (1.58) | 1003 (2.02) | 827 (2.45) | 693 (2.93) |
| C2 | 2039 | 1627 (1.25) | 1293 (1.58) | 1029 (1.98) | 833 (2.45) | 695 (2.93) |
| C3 | 2033 | 1617 (1.26) | 1287 (1.58) | 1119 (1.82) | 835 (2.43) | 723 (2.81) |
| C4 | 2039 | 1627 (1.25) | 1293 (1.58) | 1029 (1.98) | 835 (2.44) | 695 (2.93) |

Table 15: Memory usage of the Squeeze Net architecture using different input sizes on different modules over Kitti 2012 dataset

| Memory Usage in Mb (Reduction Factor) | | | | | | |
|---|---|---|---|---|---|---|
| | Input Size | | | | | |
| Conf. | 100% | 90% | 80% | 70% | 60% | 50% |
| C1 | 2031 | 1617 (1.26) | 1389 (1.46) | 1241 (1.64) | 1145 (1.77) | 893 (2.27) |
| C2 | 2041 | 1631 (1.25) | 1389 (1.47) | 1159 (1.76) | 1007 (2.03) | 877 (2.33) |
| C3 | 2035 | 1631 (1.25) | 1389 (1.47) | 1249 (1.63) | 1145 (1.78) | 893 (2.28) |
| C4 | 2041 | 1631 (1.25) | 1389 (1.47) | 1173 (1.74) | 1045 (1.95) | 877 (2.33) |

Table 16: Memory usage of the ZBontar Fast architecture using different input sizes on different modules over Kitti 2015 dataset

| Memory Usage in Mb (Reduction Factor) | | | | | | |
|---|---|---|---|---|---|---|
| | Input Size | | | | | |
| Conf. | 100% | 90% | 80% | 70% | 60% | 50% |
| C1 | 2031 | 1619 (1.25) | 1285 (1.58) | 1249 (1.58) | 833 (2.44) | 763 (2.66) |
| C2 | 2041 | 1629 (1.25) | 1295 (1.58) | 1031 (1.98) | 835 (2.44) | 697 (2.93) |
| C3 | 2035 | 1619 (1.26) | 1289 (1.58) | 1249 (1.63) | 837 (2.43) | 771 (2.64) |
| C4 | 2041 | 1629 (1.25) | 1295 (1.58) | 1029 (1.98) | 837 (2.44) | 697 (2.93) |

Table 17: Memory usage of the Squeeze Net architecture using different input sizes on different modules over Kitti 2015 dataset

| Memory Usage in Mb (Reduction Factor) | | | | | | |
|---|---|---|---|---|---|---|
| | Input Size | | | | | |
| Conf. | 100% | 90% | 80% | 70% | 60% | 50% |
| C1 | 2031 | 1631 (1.25) | 1389 (1.46) | 1303 (1.56) | 1151 (1.76) | 893 (2.27) |
| C2 | 2041 | 1631 (1.25) | 1389 (1.47) | 1173 (1.74) | 1045 (1.95) | 851 (2.40) |
| C3 | 2035 | 1631 (1.25) | 1389 (1.47) | 1303 (1.56) | 1151 (1.77) | 919 (2.21) |
| C4 | 2039 | 1631 (1.25) | 1389 (1.47) | 1185 (1.72) | 1045 (1.95) | 877 (2.32) |

### 4.0.8 Erode-Dilate module results

With the purpose of testing the efficiency of the Erode-Dilate module, new system configurations were added to the ZBontar et al. and Squeeze Net architectures. These configurations can be observed in the table 18.

Table 18: New configurations with the proposed Erode-Dilate module

| Configuration | Cost Calculation | Post-Processing | Erode-Dilate | Refinement |
|---|---|---|---|---|
| ED1 | ● | | ● | |
| ED2 | ● | | ● | ● |
| ED3 | ● | ● | ● | ● |

In this experiments the systems were tested with full sized images in order to see the direct impact of the new module on the execution time and accuracy. The GPU consumption was not taken into consideration since it's depreciable compared with the cost calculation module of the system. Also, the threshold disparity $t$ for the disparity mask process was set to 25, denoting any disparity below this value as background. The results are depicted in the table 19 and table 20 for Kitti 2012 and 2015 respectively.

The experiments shown that the Erode-Dilate module over both datasets was able to improve the accuracy of the solution with in the configurations ED1 and ED2 adding approximately only 0.01 seconds to the execution time. However, the module is not recommended with the configuration ED3 as it added errors compared with the configuration C4.

Table 19: Erode-Dilate module accuracy and execution time comparison with standard configurations over Kitti 2012 dataset

| | Execution Time in seconds (Difference with C1) | | Accuracy Error (Difference with Previous Conf.) | |
|---|---|---|---|---|
| Conf. | Squeeze Net | ZBontar Fast | Squeeze Net | ZBontar Fast |
| C1 | 0.57 sec (-) | 0.33 sec (-) | 28.77% (-) | 15.33% (-) |
| ED1 | 0.58 sec (0.01 sec) | 0.34 sec (0.01 sec) | 23.59% (5.18%) | 13.52% (1.81%) |
| C2 | 0.58 sec (0.01 sec) | 0.34 sec (0.01 sec) | 11.87% (11.73%) | 8.39% (5.12%) |
| ED2 | 0.59 sec (0.02 sec) | 0.35 sec (0.02 sec) | 9.64% (2.22%) | 7.70% (0.69%) |
| C4 | 0.90 sec (0.33 sec) | 0.66 sec (0.33 sec) | 3.69% (5.95%) | 2.93% (4.77%) |
| ED3 | 0.92 sec (0.35 sec) | 0.68 sec (0.35 sec) | 3.73% (-0.04%) | 2.94% (-0.01%) |

Table 20: Erode-Dilate module accuracy and execution time comparison with standard configurations over Kitti 2015 dataset

| | Execution Time in seconds (Difference with C1) | | Accuracy Error (Difference with Previous Conf.) | |
|---|---|---|---|---|
| Conf. | Squeeze Net | ZBontar Fast | Squeeze Net | ZBontar Fast |
| C1 | 0.59 sec (-) | 0.33 sec (-) | 28.73% (-) | 15.24% (-) |
| ED1 | 0.59 sec (-) | 0.34 sec (0.01 sec) | 24.20% (4.53%) | 13.60% (1.64%) |
| C2 | 0.59 sec (-) | 0.33 sec (-) | 12.05% (12.15%) | 8.17% (5.43%) |
| ED2 | 0.60 sec (0.01 sec) | 0.35 sec (0.02 sec) | 10.18% (1.87%) | 7.79% (0.38%) |
| C4 | 1.18 sec (0.59 sec) | 0.94 sec (0.61 sec) | 5.05% (5.13%) | 4.04% (3.75%) |
| ED3 | 1.20 sec (0.61 sec) | 0.95 sec (0.62 sec) | 5.06% (-0.01%) | 4.05% (-0.01%) |

## 5 Conclusions and Future Work

The long term objective of this research is to reduce the computational resources required by these models, to make their deployment on smaller devices feasible. These resources include memory, execution time, storage and communications size. In this paper we presented a Cost Calculation CNN based module built as a Squeeze Network to generate an initial disparity map. This network was combined with post-processing and refinement algorithms to improve the final disparity map quality. In the tests performed the system showed a reduction of almost 30% of the number of parameters. The cost of such a reduction was less than 1.5% of accuracy and less than 250 ms of execution time when compared to state of the art networks. The GPU memory used was comparable with the fast architecture and consumed less than a half the accurate architecture. Thus, the results show the utility of our architecture in terms of reducing the size of the network, and it is a first step towards the more general goal of reducing the execution time and memory required.

The model proposed might be improved to obtain better quality on disparity maps and less execution time by experimenting with different architectures and hyperparameters, like the combination of squeeze nets and residual networks. Also, different techniques of parameter size reduction like pruning, as mentioned in Iandola et al[15], to minimize even more the network size, will be explored in future work.

In addition, the 50% reduction of the input parameters shown a decrement of 4.36 and 3.39 times the execution time using Kitti 2012, or 5.52 and 4.14 times the execution time using Kitti 2015 in the ZBontar et al. and Squeeze Net architectures respectively with a cost of only 1.31 and 1.56 times increment of the accuracy error in the C3 architectures configuration using Kitti 2012, or 1.41 and 1.64 times using Kitti 2015, denoting the potential of such technique to make these architectures closer to real time applications. In other configurations like C1, this technique not only reduced the execution time but also improved the accuracy of the system reaching a 5.37% and 5.4% less error rate and a 66.6% and 62.1% reduction on the execution time using Kitti 2012. In our tests using Kitti 2015 the same configuration reached 4.03% and 6.77% less error rate and similar reduction on the execution time. In all the cases, the reduction of the input size helped to consume less GPU memory making it feasible to use on smaller devices with less resources. We believe that this technique in combination with parameter size reduction, can help even more to decrement the amount of GPU memory required by these systems.

Finally, taking into consideration the execution cost of the post-processing module, the Erode-Dilate module offered a low execution cost alternative to improve the solution quality when the post-processing module is not used.

## 6 Acknowledgements

This research was supported by CONICET (National Council of Scientific and Technological Research), and LABSIN (Intelligent Systems Laboratory) of the School of Engineering (National University of Cuyo). We also want to gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

## References

[1] Hannah, M.J.: Computer matching of areas in stereo images. Technical, Stanford University (1974)

[2] Marr, D., Poggio, T.: Cooperative computation of stereo disparity. Science **194**(4262), 283–287 (1976)

[3] Scharstein, D., Szeliski, R.: A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. International Journal of Computer Vision **47**(1–3), 7–42 (2002)

[4] Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**(6088), 533–536 (1986)

[5] LeCun, Y., Boser, B.E., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.E., Jackel, L.D.: Backpropagation Applied to Handwritten Zip Code Recognition. Neural Computation **1**(4), 541–551 (1989)

[6] Hinton, G.E.: Learning multiple layers of representation. Trends in Cognitive Sciences **11**(10), 428–434 (2007)

[7] Torch home page, http://torch.ch/

[8] Tensorflow home page, https://www.tensorflow.org/

[9] Theano home page, http://deeplearning.net/software/theano/

[10] Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems **25**(2), 1097–1105 (may 2012)

[11] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015), http://arxiv.org/abs/1512.03385

[12] Žbontar, J., LeCun, Y.: Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches. Journal of Machine Learning Research **17**, 1–32 (2016)

[13] ImageNet Large Scale Visual Recognition Challenge (ILSVRC) home page, http://image-net.org/challenges/LSVRC/

[14] The KITTI Vision Benchmark Suite evaluation page, http://www.cvlibs.net/datasets/kitti/eval_stereo.php

[15] Iandola, F.N., Moskewicz, M.W., Ashraf, K., Han, S., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. CoRR **abs/1602.07360** (2016), http://arxiv.org/abs/1602.07360

[16] Hamzah, R.A., Ibrahim, H.: Literature Survey on Stereo Vision Disparity Map Algorithms. Journal of Sensors **2016**, 1–23 (2016)

[17] Zeiler, M.D., Fergus, R.: Visualizing and Understanding Convolutional Networks. Computer Vision-ECCV 2014 **8689**, 818–833 (2014)

[18] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision **115**(3), 211–252 (2015)

[19] Mei, X., Sun, X., Zhou, M., Jiao, S., Wang, H., Xiaopeng Zhang: On building an accurate stereo matching system on graphics hardware. In: 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops). pp. 467–474. IEEE, Barcelona (nov 2011)

[20] Shaked, A., Wolf, L.: Improved stereo matching with constant highway networks and reflective confidence learning. CoRR **abs/1701.00165** (2017), http://arxiv.org/abs/1701.00165

[21] Yang, M., Lv, X.: Learning both matching cost and smoothness constraint for stereo matching. Neurocomputing **314**, 234 – 241 (2018)

[22] Ke Zhang, Jiangbo Lu, Lafruit, G.: Cross-Based Local Stereo Matching Using Orthogonal Integral Images. IEEE Transactions on Circuits and Systems for Video Technology **19**(7), 1073–1079 (jul 2009)

[23] Hirschmüller, H.: Stereo Processing by Semiglobal Matching and Mutual Information. IEEE Transactions on Pattern Analysis and Machine Intelligence **30**(2), 328–341 (feb 2008)

[24] Yao, G., Liu, Y., Lei, B., Ren, D.: A rapid stereo matching algorithm based on disparity interpolation. In: Proceedings of 2010 Conference on Dependable Computing. pp. 5–10 (2010)

[25] Williem, W., Kyu Park, I.: Deep self-guided cost aggregation for stereo matching. Pattern Recognition Letters **112** (07 2018)

[26] Miclea, V.C., Vancea, C.C., Nedevschi, S.: New sub-pixel interpolation functions for accurate real-time stereo-matching algorithms. In: 2015 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP). vol. 20, pp. 173–178. IEEE (sep 2015)

[27] Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In: Conference on Computer Vision and Pattern Recognition (CVPR) (2012)

[28] Menze, M., Heipke, C., Geiger, A.: Joint 3d estimation of vehicles and scene flow. In: ISPRS Workshop on Image Sequence Analysis (ISA) (2015)

[29] Menze, M., Heipke, C., Geiger, A.: Object scene flow. ISPRS Journal of Photogrammetry and Remote Sensing (JPRS) (2018)